IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2019

MSc in Computing Science
for Internal Students of the Imperial College of Science, Technology and Medicine

PAPER M1

(517+518) PROGRAM DESIGN AND LOGIC

Tuesday 30th April 2019, 10:00
Duration: 120 minutes

Answer THREE questions

Paper contains 4 questions
Calculators not required

Section A *(Use a separate answer book for this Section)*

1 You are tasked with implementing a web page that collects news items from multiple sources (e.g., news agencies, blogs, companies issuing press releases, etc.), files them into categories (e.g., "technology", "politics" or "cooking") and presents them to users.

Each news item has a title, content and is attributed to an author (identified by their name). Each news item has a counter representing how often an item has been viewed. Each news item is owned by exactly one source. A source can own many news items. A source has a unique URL/web address (i.e., there are no two different sources with the same URL). Each news item is associated with exactly one category. Categories, once created, cannot be deleted. A category can (but need not) have a parent category.

For simplicity, you can assume that there are no more than 100 sources, no more than 1000 news items per source and no more than 500 categories in total.

Hint: Objects (i.e., instances of classes) that are not owned by other objects shall be accessible through a global variable.

a Implement the classes in C++ in the file `newsaggregator.cpp`. Select appropriate types for the class members and implement the necessary constructors and destructors. You may **not** use smart pointers.

b Implement a function that creates a new news item. The use of the function is illustrated by the following snippet:

```
auto title = "Something Happened";
auto text = "London is a city where things happen all the time";
auto author = "A random stranger";
auto categoryName = "London News";
auto sourceWebpage = "http://www.blameberg.com";

insertNewsItem(title, text, author, categoryName, sourceWebpage);
```

You can assume that a category with that name ("London News") already exists. Feel free to define utility functions and reuse them in later parts of this question.

c Implement a function that gets the content of a news item (identified by the url of its source and its title) and increases its click count. The use of the function is illustrated by the following snippet:

```
auto sourceWebpage = "http://www.blameberg.com";
auto title = "Something Happened";

string content = getContent(sourceWebpage, title);
```

Feel free to define utility functions and reuse them in later parts of this question.

d Implement a function that finds the hottest category, defined as the one with the most clicked news items. The use of the function is illustrated by the following snippet:

```
auto categoryName = getHottestCategory();
```

You may use the stl class template `std::map` (see declaration in `newsaggregator.cpp`).

The four parts carry, respectively, 30%, 25%, 15%, and 30% of the marks.

2 Consider the following scenario related to classifying fruits that are to be sold:

- A fruit has a weight (in grams), a best before date (in days since today, i.e. today is 0, tomorrow is 1, etc.), and a name of the variety of the fruit (see table under 2c for examples of names of varieties). A fruit is considered fit for sale if there are at least 3 days left until the best before date has been reached.

- Apples and pears are fruits and they can be classified and sold as 'Premium Class' if they meet the requirements dictated by the Specific Marketing Standards:

Type of Fruit	Minimum Requirements
All Types of Apples	weight of 90g
Red Apples	75% of total surface is red coloured
Mixed Apples	50% of total surface is red coloured
Pear	the stalk must be intact and a weight of 130g

As can be seen in the table above, apples can be divided into 2 different types depending on their skin colour when they are ripe: red apples with mostly red skin and mixed apples with a mix of red and green skin.

- A fruit basket can hold a maximum number of fruits, which can be individually added to or removed from the basket. At any time the contents of the fruit basket can be listed by printing the name of the variety, weight, and best before date of every fruit in the basket to standard output.

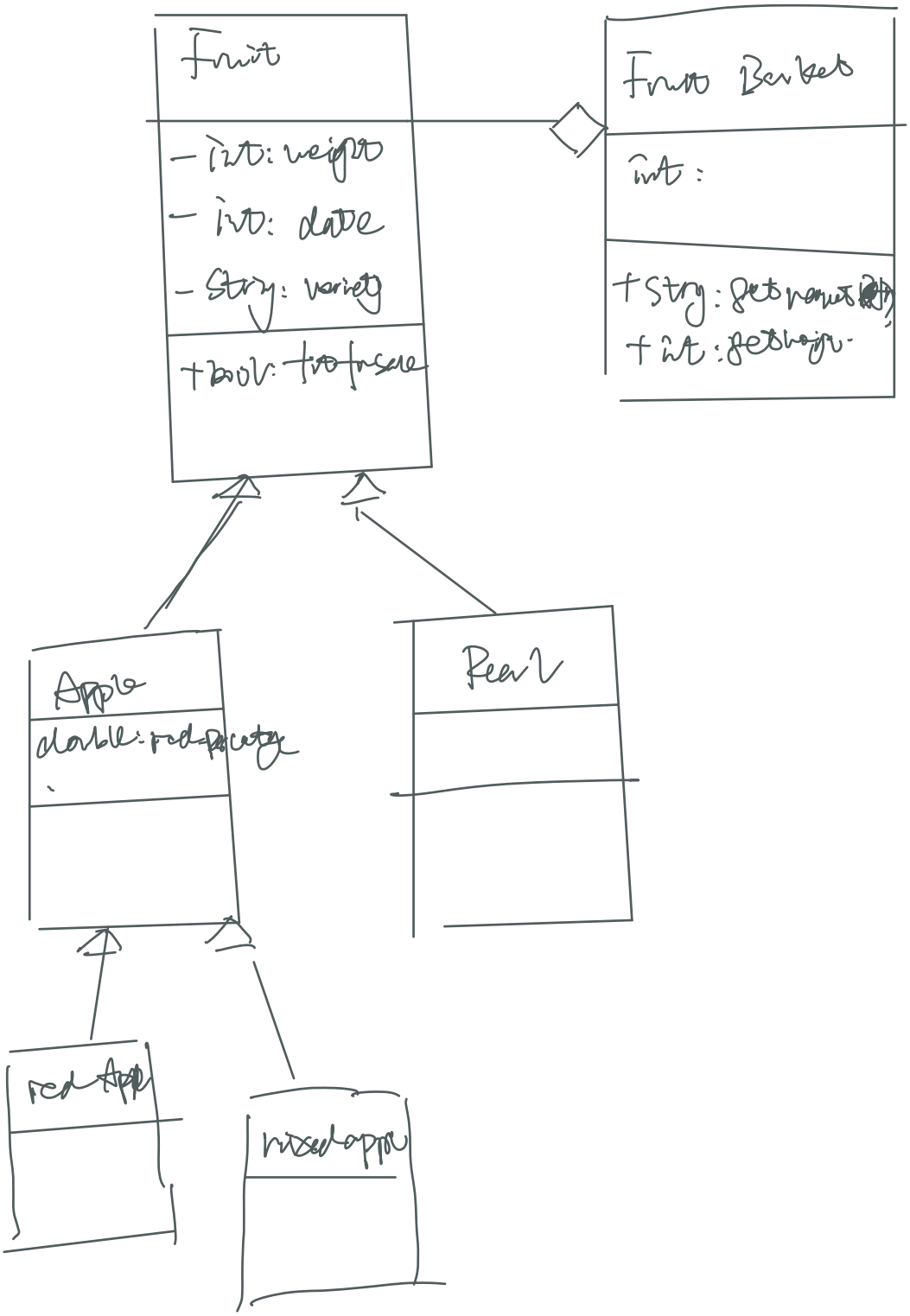
- Use the template class *set* as the underlying container. The relevant declarations can be found in the provided skeleton file *answer.cpp*.

a Draw a UML class diagram that describes the public interfaces and associations of the above. Use the provided answer book for this part of the question.

b Write C++ class declarations and definitions to support the above in the file *answer.cpp*. Ensure that the fruit basket is instantiated from a class template and that the number of items a basked can hold is specified by a template parameter. The class template should make it possible to create different types of baskets (i.e. not only fruit baskets) that can hold different types of object.

c Add a test function to *answer.cpp* that does the following:

Create a fruit basket that can hold 30 fruits. Create the following fruits and add them all to the basket before printing the contents of the basket:


Fruit
- int: weight
- int: date
- String: variety
+bool: isFresh
Fruit Basket
int:
+String: getVariety()
+int: getWeight()
Apple
double: redPercentage
Pear
redApple
mixedApple

Type of Fruit	Variety	Weight	Best Before	Red Surface	Intact Stalk
Red Apple	Gala	95g	7 days	80%	N/A
Mixed Apple	Cox	80g	8 days	60%	N/A
Pear	Ambrosia	140g	2 days	N/A	Yes

Then create a fruit basked that can only hold 10 fruits. Take one fruit from the bigger basket at a time and either (a) add them to the smaller basket if the fruit is fit for sale and classifies as 'Premium Class' or (b) throw it away. Print the contents of both baskets.

The three parts carry, respectively, 15%, 70%, and 15% of the marks.

Section B *(Use a separate answer book for this Section)*

Note: All natural deduction proofs must be presented clearly, with wff numbering, where appropriate, indentations, and explanations. Marks will be deducted for unclear and poorly presented proofs. When using natural deduction, unless otherwise stated, you may use any of the primitive and derived rules, but not equivalences, unless they are proved by natural deduction, themselves.

3 Consider the following sentences of logic

T1	$p \wedge \neg q \rightarrow r \vee s$	*T6*	$r \wedge v \rightarrow m$
T2	p	*T7*	$\neg v \rightarrow w$
T3	k	*T8*	$s \rightarrow m \vee n$
T4	$\neg(k \wedge q)$	*T9*	$k \rightarrow \neg n$
T5	$r \wedge w \rightarrow m$		

a Show by natural deduction that
$T1,T2,T3,T4,T5,T6,T7,T8,T9 \vdash m$.

b Consider the following sentences of logic

S1 $v \rightarrow m$
S2 $s \rightarrow m$

Show by resolution that
$T1,T2,T3,T4,T5,T7,S1,S2 \vdash m$.
State clearly the CNF of all the clauses and show the resolution. You do not need to show how the CNF of the clauses are worked out.

Reminder: A wff is in CNF if it is of the form: $W_1 \wedge W_2 \wedge .... \wedge W_n$, $n \geq 1$, and each W_i is a disjunction of literals.

c Show the following equivalence:

$\exists X,Y \ (p(X) \wedge q(X,Y) \wedge \neg r(X,Y)) \equiv \neg (\forall X,Y \ (p(X) \wedge q(X,Y) \rightarrow r(X,Y)))$

d Let S represent the sentences of logic P1-P6 below, about customers' requests to purchase items and rules for satisfying such requests (*a* is the identifier of a customer and *i* is the identifier of an item):

P1 $customer(a)$
P2 $requests(a,i)$
P3 $\neg inStore(i)$
P4 $\neg inStock(i)$
P5 $\forall C,I \ (satisfyRequest(C, I) \leftrightarrow requests(C, I) \wedge (inStore(I) \vee inWarehouse(I)))$
P6 $\forall I \ (\neg inStore(I) \rightarrow (inWarehouse(I) \rightarrow inStock(I)))$

Show that
$S \vdash \neg (\forall C,I \ (customer(C) \wedge requests(C,I) \rightarrow satisfyRequest(C, I)))$
using natural deduction and the equivalence in (c) if required.

Parts a, b, c, d carry 30%, 30%, 15%, 25% of the marks, respectively.

4 Formalise in predicate logic the sentences (i)-(vi) below, that concern the rules of a system of voting. Use only the predicates listed below and the infix predicates $=, <, >, \leq, \geq$ if required. Ensure that you present your formulas clearly, using brackets to correctly identify the scope of quantifiers and disambiguate where necessary.

vote(X,D,V)	*to mean X votes on day D for outcome V.*
eligible(X)	*to mean X is eligible to vote.*
open(D)	*to mean voting opens on day D.*
permitted(X, D)	*to mean X is permitted to vote on day D.*
fined(X)	*to mean X is fined.*
unanimous(V)	*to mean the vote for outcome V is unanimous.*
legChall(V)	*to mean there is legal challenge against outcome V.*
binding(V)	*to mean outcome V is binding.*

i) A vote at any time can be either a 'yes' or a 'no'.

ii) Anyone eligible to vote is permitted to vote within 2 days of the opening of voting.

iii) No eligible person can vote more than once (that is they can vote only on one day and for one outcome).

iv) Anyone eligible to vote who does not vote within 2 days of voting opening is fined.

v) If at least one vote is cast and all votes are for the same outcome, then that outcome is unanimous.

vi) A unanimous outcome is binding unless there is a legal challenge against it.

Parts (i) - (vi) carry 10%, 20%, 15%, 20%, 25%, 10% of the marks, respectively.

```
/* Add ALL of your code to this file. */

#include <string>
#include <utility>
using namespace std;

namespace std {

template <class Key, // map::key_type
          class T>   // map::mapped_type
class map {
public:
  typedef std::pair<const Key, T> value_type;
  typedef value_type* iterator;

  /**
   * Returns an iterator referring to the first element in the map
   * container.
   */
  iterator begin();

  /**
   * Returns an iterator referring to the past-the-end element in the map
   * container.
   */
  iterator end() const;

  /**
   * If k matches the key of an element in the container, the function
   * returns a reference to its mapped value.
   *
   * If k does not match the key of any element in the container, the
   * function inserts a new element with that key and returns a
   * reference to its mapped value. Notice that this always increases
   * the container size by one, even if no mapped value is assigned to
   * the element (the element is constructed using its default
   * constructor).
   */
  T& operator[](const Key& k);

  /**
   * Extends the container by inserting new elements, effectively
   * increasing the container size by the number of elements inserted.
   *
   * Because element keys in a map are unique, the insertion operation
   * checks whether each inserted element has a key equivalent to the
   * one of an element already in the container, and if so, the
   * element is not inserted, returning an iterator to this existing
   * element (if the function returns a value).
   */
  pair<iterator, bool> insert(const value_type& val);

  /**
   * Returns a reference to the mapped value of the element identified
   * with key k.
   *
   * If k does not match the key of any element in the container, the
   * function throws an out_of_range exception.
   */
  T& at(const Key& k);

  /**
   * Searches the container for elements with a key equivalent to k and
   * returns the number of matches.
   *
```

```
   * Because all elements in a map container are unique, the function
   * can only return 1 (if the element is found) or zero (otherwise).
   */
  size_t count(const Key& k) const;
};
} // namespace std

/* Add your code below this line. */

// Question a

// Question b

// Question c

// Question d

/* Here are examples of how the functions should be used (copied from
   exam paper). */

/* DO NOT CHANGE ANYTHING BELOW THIS LINE!!! */

void usage1() {
  auto title = "Something Happened";
  auto text = "London is a city where things happen all the time";
  auto author = "A random stranger";
  auto categoryName = "London News";
  auto sourceWebpage = "http://www.blameberg.com";

  insertNewsItem(title, text, author, categoryName, sourceWebpage);
}

void usage2() {
  auto sourceWebpage = "http://www.blameberg.com";
  auto title = "Something Happened";

  string content = getContent(sourceWebpage, title);
}

void usage3() { //
  auto categoryName = getHottestCategory();
}

int main(int argc, char* argv[]) {
  usage1();
  usage2();
  usage3();
  return 0;
}
```

```
/* Add ALL of your code to this file. */

/* The declarations of the class template 'set' are below. Do NOT uncomment them.

template <typename T> class set {
  public:
    set(); // constructor that creates an empty set
    void insert(const T& item); // adds item to the set
    set<T>::constant_iterator begin(); // returns constant iterator
    set<T>::constant_iterator end(); // returns constant iterator
    void erase(set<T>::const_iterator); // removes specified item
    bool empty(); // returns true if the set is empty
};

*/

/* Do NOT add any other header files. */

#include<iostream>
#include<string>
#include<set>

/* Add your code below this line. */

// Question b

// Question c

int main() {

}
```